Light mapping in a nutshell

Paul Nettle, June 21, 1999

In this document, I'll attempt to explain the basic process of light mapping. I won't dive into the intricacies of how the lighting information is calculated (i.e. via ray tracing, radiosity, etc.) Rather, I'll explain how to use light mapping to your advantage and how to apply these concepts in a real-world application.

The concept of light mapping is pretty simple. The illumination across a surface is rendered (usually pre-rendered) into a texture. When that texture is then applied to a given surface in the appropriate way, you get the illusion of illumination. This results in a more aesthetically pleasing scene without the performance cost of the actual lighting calculations, but with the cost of static pre-calculated lighting.

The first steps

We'll start with a few assumptions. First, that you're targeting a platform with 3D acceleration, and second, that you've already got your scene loaded up and ready to play with.

We'll need some extra data to manage our light maps. We'll need an ID to each surface (i.e. polygon) in the scene. We'll need this because there will be just as many textures as we have polygons since they each may be illuminated uniquely. Along with this ID, we'll need to store a new set of texture coordinates for each polygon. This way, the polygon can have a texture applied to it, and a light-map applied to it. By storing unique UV values for textures and light-maps, we allow each to be mapped independently. If you're planning to support software rendering, this might not be such a great idea, but this is beyond the scope of this document (don't you just hate it when people write stuff like that?)

The light-map UVs

We'll need to pick the light-map UVs for each surface. This is a pretty simple process, but before I get into this, let's discuss a few small details.

Texel density is very important. This refers to how tightly packed the texels are applied to the surface. For example, a one-square-foot surface with a 256x256 texture applied to it will have a much higher texel density than a the same surface with only a 32x32 applied to it.

Ideally, we would like to have a uniform texel density across the entire scene (at least, to start with…)

Aside from texel density, we need to consider polygon seams (where two adjoining polygons meet along a shared edge.) These shared-edged polygons should have similarly mapped light maps so that the seam is not noticeable. This usually applies to the standard texture maps, but it is more important for light maps. Even non-planar polygons should have a shared mapping across the bend where they meet to keep the light continuous.

The only way to accomplish this is to maintain the same texel density as well as the same mapping scheme.

Another thing we want to try to accomplish is to have an even resolution across a surface. Ideally, we want to minimize the "stretch" of a light-map across any surface, giving us as close to an equal resolution in U as in V.

The mapping scheme

To accomplish this seamless mapping across planar as well as non-planar surfaces, we'll need a shared frame of reference (a frame of reference that all polygons fit into) for all mapping. I'll use world-space coordinate system as a frame of reference for this example, but feel free to find your own.

World-space mapping is done by using two of the three world-space components (X&Y, Y&Z or Z&X) and applying them directly to the UV coordinates. By using these world-space coordinates we effectively apply a planar mapping to everything in the scene using the plane defined by the coordinates chosen. And magically, any polygons that share a common edge (and hence, vertices that define those edges) will end up with the same UV values at those edges.

To properly map a 3D scene, we can't use just one planar mapping, we need to use all three (X/Y/Z). To accomplish this, we simply consider each surface normal. For the surfaces whose normal is primarily in the X/Y plane (i.e. Z is the largest value in the normal) we'll use the X&Y world-space coordinates…

```
if (surface.normal.z > surface.normal.y && surface.normal.y > surface.normal.x)
{
     for (each vertex i)
     {
          surface.lightmap[i].u = surface.vertex[i].x;
     surface.lightmap[i].v = surface.vertex[i].y;
     }
}
if (surface.normal.y > surface.normal.x && surface.normal.x > surface.normal.z)
{
```

```
        for (each vertex i)
        {
             surface.lightmap[i].u = surface.vertex[i].z;
        surface.lightmap[i].v = surface.vertex[i].x;
        }
    }
    if (surface.normal.x > surface.normal.z && surface.normal.z > surface.normal.y)
    {
        for (each vertex i)
        {
             surface.lightmap[i].u = surface.vertex[i].y;
        surface.lightmap[i].v = surface.vertex[i].z;
        }
    }
```

In the above example code, you'll see that the planar mapping is applied specific to the surface direction to get the most resolution out of each surface. Since world-space is a 3-space, we'll get seamless edges across non-planar polygons as well as planar polygons.

You may find that world-space coordinates will give you a mapping that is not ideal in terms of resolution or texel density. You can adjust this by applying a scalar to each vertex before applying it to the UV value. For example, if your world space coordinate system is mapped to a 1:1 correlation of units to feet, and you want two texels per feet, simply multiply your light-map UV values by 2.

The resolution of the light maps is completely up to you. I will suggest that you use as high of a resolution as you can get away with. Play with it and see what you find works best.

Light-map sizes

Many 3D accelerators have limitations on their texture dimensions (power of two sizes only, etc.) and maximum resolutions (2048x2048, etc.)

If a surface requires a light-map that exceeds your 3D accelerator's maximum resolution, that surface will need to be split. Surfaces require an odd sized light-map (65x75) on an accelerator that requires power-of-two texture dimensions will end up wasting some space. This waste can be reduced, however (see "Optimizations").

Caching the textures

Caching these textures should work pretty much the same as caching the standard textures. I'm only including this section to point out one important aspect. Each surface will have its own unique light-map, which means a LOT of light maps.

Fortunately, these maps are usually much smaller than the standard textures used to give the scene its detail. This reduces the amount of memory required, but your combined light maps will probably be much larger than your combined detail texture library.

Optimizations

There really are a lot of potential optimizations. For example, if you have many small surfaces that map to the same planar mapping and share edges, these surfaces can be combined into a single light-map. This improves your ratio of light-maps to surfaces as well as reduces potential wasted light-map texture space.

You could take this a step further and place smaller textures into the wasted areas of larger textures, though this can prove to be quite tricky.

Using n-gons rather than triangles for your data can effectively help accomplish these tasks by combining small planar polygons.

This list goes on…

Further reading

I wrote a document a while back that discusses the surface caching mechanism used in KAGE. A surface cache is used for software rendering systems that store light-maps blended with their surface texture to avoid a dual-texturing pass and improve performance (thanks to John Carmack for this bit o' brilliance.)

Although KAGE has abandoned the surface cache for a more flexible design, this document still offers some good insight into the process of light-mapping.